ORACLE
Linux

FIPS 140-2 Non-Proprietary Security Policy

Oracle Linux 8 OpenSSL Cryptographic Module

FIPS 140-2 Level 1 Validation

Software Version: R8-8.4.0

Date: April 29th, 2022

Title: Oracle Linux 8 OpenSSL Cryptographic Module Security Policy
Date: April 29th, 2022
Author: Oracle Security Evaluations – Global Product Security
Contributing Authors:
Atsec Information Security
Oracle Linux Engineering

Oracle Corporation
World Headquarters
2300 Oracle Way
Austin, TX 78741
U.S.A.
Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
www.oracle.com

ORACLE

TABLE OF CONTENTS

List of Tables

List of Figures

ORACLE

1. Introduction

1.1 Overview

Oracle Linux is a set of cryptographic libraries, services, and user level cryptographic applications that are validated at FIPS 140-2 level 1, providing a secure foundation for vendor use in developing dependent services, applications, and even purpose built appliances that may be FIPS 140-2 validated.

This section is informative to the reader to reference other cryptographic services of Oracle Linux. Only the software listed in section 3.1 is subject to the FIPS 140-2 validation. The CMVP makes no statement as to the correct operation of the module or the security strengths of the generated keys when supported if the specific operational environment is not listed on the validation certificate.

The FIPS 140-2 validation is performed at Security Level 1, on software only modules that do not make any claims about the hardware enclosure. This allows vendors to develop their own modules using these basic cryptographic modules and validate the new modules at a higher FIPS 140-2 security level.

The following cryptographic modules are included in Oracle Linux 8:

- OpenSSL– a software cryptographic module supporting FIPS 140-2-approved cryptographic algorithms for protocols like TLS 1.2, TLS 1.3, OpenSSH, and HTTPS
- NSS Softokn Cryptographic Module – supplies cryptographic support for TLS, PKCS #5, PKCS #7, PKCS #11 (version 3.0), PKCS #12, S/MIME, X.509 v3 certificates, and other security standards supporting FIPS 140-2 validated cryptographic algorithms
- Oracle Linux Unbreakable Enterprise Kernel (UEK 6) – a software only cryptographic module via the Kernel Crypto API that is an optimized kernel with a wide range of advanced features and improvements for enterprise workloads. The UEK provides general-purpose cryptographic services and block storage encryption.
- GnuTLS – general purpose cryptographic module to support TLS network protocols
- Libgcrypt – supplies general cryptographic support for GPG.

This Security Policy describes the features and design of the Oracle Linux OpenSSL Cryptographic Module using the terminology contained in the FIPS 140-2 specification. FIPS 140-2, Security Requirements for Cryptographic Module specifies the security requirements that will be satisfied by a cryptographic module utilized within a security system protecting sensitive but unclassified information. The NIST/CSE Cryptographic Module Validation Program (CMVP) validates cryptographic module to FIPS 140-2. Validated products are accepted by the Federal agencies of both the USA and Canada for the protection of sensitive or designated information.

1.2 Document Organization

The Security Policy document is one document in a FIPS 140-2 Submission Package. In addition to this document, the Submission Package contains:

- Oracle Linux 8 OpenSSL Cryptographic Module Non-Proprietary Security Policy
- Other supporting documentation as additional references

With the exception of this Non-Proprietary Security Policy, the FIPS 140-2 Validation Documentation is proprietary to Oracle and is releasable only under appropriate non-disclosure agreements. For access to these documents, please contact Oracle.

2. Oracle Linux 8 OpenSSL Cryptographic Module

2.1 Functional Overview

The Oracle Linux 8 OpenSSL Cryptographic Module (hereafter referred to as the "Module") is a software module supporting FIPS 140-2 Approved cryptographic algorithms within Oracle Linux. The code base of the Module is formed in a combination of standard OpenSSL shared Library and development work by Oracle Linux engineering. The scope of testing for this validation covers version R8-8.4.0 running Oracle Linux 8.4. The Oracle Linux OpenSSL Module is distributed with an open-source distribution. The Module provides a C language application program interface (API) for use by other processes that require cryptographic functionality.

Oracle Linux 8 OpenSSL supports following three types of cryptographic implementations. The implementations available for an algorithm are listed in Table 2 and they can be selected based on the environment variable OPENSSL_ia32cap. Please refer to its man page for the details.

a) OpenSSL in Native C Programming Language;
b) AES-NI hardware acceleration for X86 processors;
c) AES optimizations for ARM processors
d) Assembler implementation.

2.2 FIPS 140-2 Validation Scope

The following table shows the security level for each of the eleven sections of the validation. See Table 1 below.

Security Requirements Section	Level
Cryptographic Module Specification	1
Cryptographic Module Ports and Interfaces	1
Roles and Services and Authentication	1
Finite State Machine Model	1
Physical Security	N/A
Operational Environment	1
Cryptographic Key Management	1
EMI/EMC	1
Self-Tests	1
Design Assurance	3
Mitigation of Other Attacks	1

Table 1: FIPS 140-2 Security Requirements

3. Cryptographic Module Specification

3.1 Definition of the Cryptographic Module

The Oracle Linux OpenSSL Module is defined as a multi-chip standalone module as defined by the requirements within FIPS PUB 140-2. The logical cryptographic boundary of the module consists of shared library files and their integrity check HMAC files, which are delivered through the Oracle Linux Yum Server as listed below:

The module version R8-8.4.0 was tested on Oracle Linux 8.4 and is contained within the RPM file with version openssl-libs-1.1.1g-15.el8_3.x86_64.rpm or openssl-libs-1.1.1g-15.el8_3.aarch64.rpm, which contains the following files:

- /usr/lib64/.libcrypto.so.1.1.1g.hmac (64 bits)
- /usr/lib64/.libssl.so.1.1.1g.hmac (64 bits)
- /usr/lib64/libcrypto.so.1.1.1g (64 bits)
- /usr/lib64/libssl.so.1.1.1g (64 bits)

The Oracle OpenSSL package includes the binary files, integrity check HMAC files, Man Pages and the OpenSSL engines provided by the standard OpenSSL shared library. The OpenSSL engines and their shared object files are not part of the Module, and therefore they must not be used when operating the Module.

Figure 1 shows the logical block diagram of the module executing in memory on the host system.

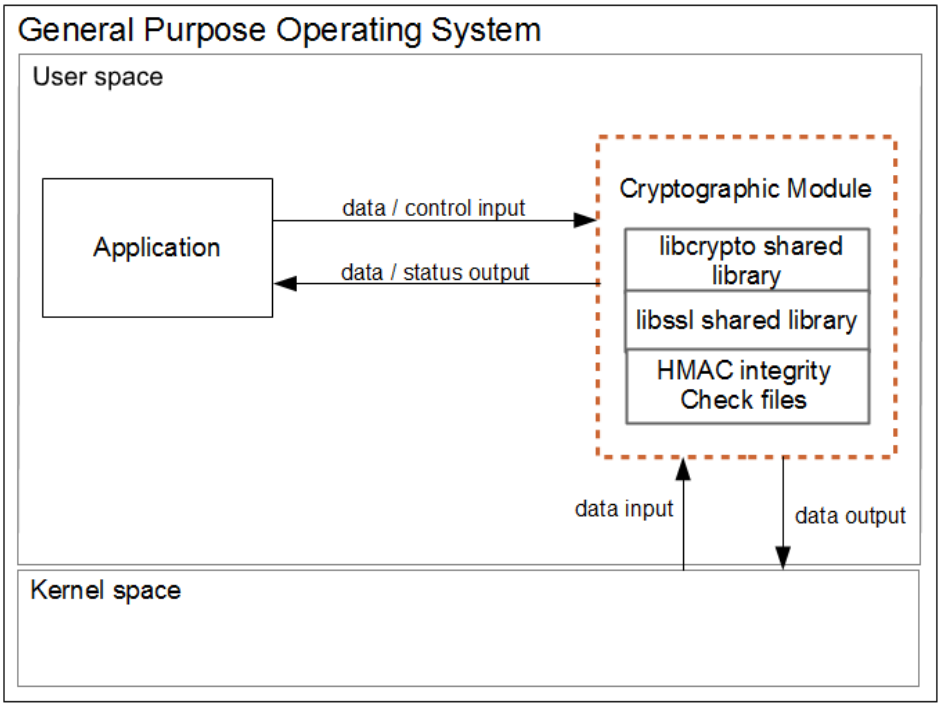

Figure 1: Oracle Linux OpenSSL Logical Cryptographic Boundary

3.2 Definition of the Physical Cryptographic Boundary

The physical cryptographic boundary of the module is defined as the hard enclosure of the host system on which it runs. See figure 2 below. No components are excluded from the requirements of FIPS PUB 140-2.

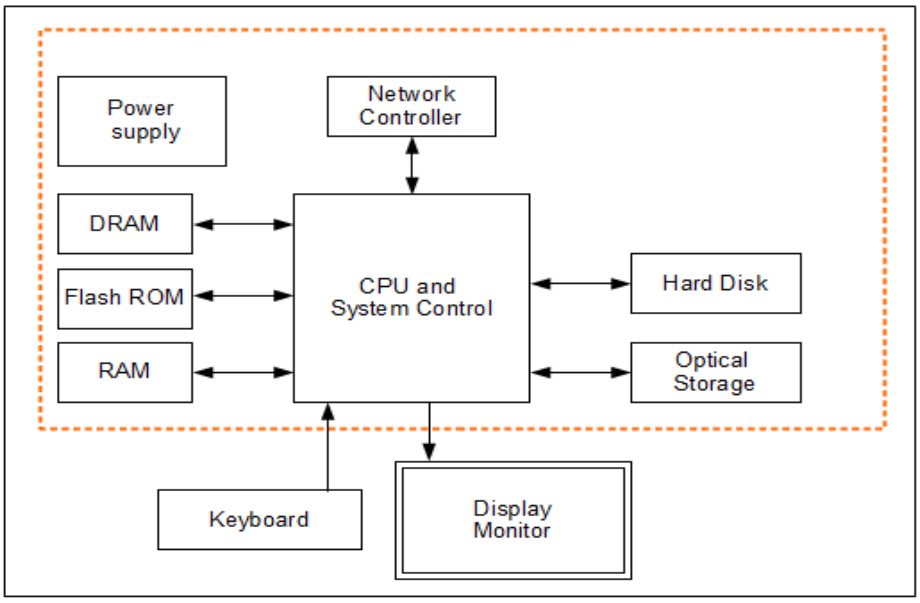

Figure 2: Oracle Linux OpenSSL Hardware Block Diagram

3.3 Approved or Allowed Security Functions

The Oracle Linux OpenSSL Cryptographic Module contains the following FIPS Approved Algorithms listed in Table 2. Note that not all algorithms/modes tested with a corresponding CAVP cert are implemented/used by the module.

Once the module is operational, the mode of operation is implicitly assumed depending on the services/security function invoked. By default, the module enters Approved mode after the power-up tests succeed. In Approved mode, only approved or allowed security functions can be used as specified in table 2 and 3 and services listed in table 8.

Approved or Allowed Security Functions		Cert #
Symmetric Algorithms		
AES	**AESNI:** AES in CBC, ECB, CFB1, CFB8, CFB128, OFB, CTR, CCM, CMAC, KW, KWP, XTS Modes (Key Sizes 128, 192, 256 for all modes except XTS Mode where key sizes are 128 and 256)	A 1673
	AESASM: AES in CBC, ECB, CFB1, CFB8, CFB128, OFB, CTR, CCM, CMAC, KW, KWP, XTS Modes (Key Sizes 128, 192, 256 for all modes except XTS Mode where key sizes are 128 and 256)	A 1674
	BAES_CTASM: AES in CBC, ECB, CFB1, CFB8, CFB128, OFB, CTR, CCM, CMAC, KW, KWP, XTS Modes (Key Sizes 128, 192, 256 for all modes except XTS Mode where key sizes are 128 and 256)	A 1675

Approved or Allowed Security Functions		Cert #
	SSH_AVX2: AES in ECB mode (Key Sizes 128, 192, 256)	A 1677
	SSH_AVX: AES in ECB mode (Key Sizes 128, 192, 256)	A 1678
	SSH_SSSE3: AES in ECB mode (Key Sizes 128, 192, 256)	A 1679
	SSH_ASM: AES in ECB mode (Key Sizes 128, 192, 256)	A 1680
	AESNI_AVX: AES in GCM and GMAC modes (Key sizes 128, 192, 256)	A 1685
	AESNI_CLMULNI: AES in GCM and GMAC modes (Key sizes 128, 192, 256)	A 1686
	AESNI_ASM: AES in GCM and GMAC modes (Key sizes 128, 192, 256)	A 1687
	AESASM_AVX: AES in GCM and GMAC modes (Key sizes 128, 192, 256)	A 1688
	AESASM_CLMULNI: AES in GCM and GMAC modes (Key sizes 128, 192, 256)	A 1689
	AESASM_ASM: AES in GCM and GMAC modes (Key sizes 128, 192, 256)	A 1690
	BAES_CTASM_AVX: AES in GCM and GMAC modes (Key sizes 128, 192, 256)	A 1691
	BAES_CTASM_CLMULNI: AES in GCM and GMAC modes (Key sizes 128, 192, 256)	A 1692
	BAES_CTASM_ASM: AES in GCM and GMAC modes (Key sizes 128, 192, 256)	A 1693
	AES_C: AES in CBC, ECB, CFB1, CFB8, CFB128, OFB, CTR, CCM, CMAC, KW, KWP, XTS Modes (Key Sizes 128, 192, 256 for all modes except XTS Mode where key sizes are 128 and 256)	A 2139
	CE: AES in CBC, ECB, CFB1, CFB8, CFB128, OFB, CTR, CCM, CMAC, KW, KWP, XTS Modes (Key Sizes 128, 192, 256 for all modes except XTS Mode where key sizes are 128 and 256)	A 2141
	VPAES: AES in CBC, ECB, CFB1, CFB8, CFB128, OFB, CTR, CCM, CMAC, KW, KWP, XTS Modes (Key Sizes 128, 192, 256 for all modes except XTS Mode where key sizes are 128 and 256)	A 2142
	AES_C_GCM: AES in GCM and GMAC modes (Key sizes 128, 192, 256)	A 2144
	CE_GCM: AES in GCM and GMAC modes (Key sizes 128, 192, 256)	A 2145
	VPAES_GCM: AES in GCM and GMAC modes (Key sizes 128, 192, 256)	A 2146

Approved or Allowed Security Functions		Cert #
Triple DES	**TDES_C:** CBC, ECB, CFB1, CFB8, CFB64, OFB, CMAC Modes (d/e; KO 1)	A 1672
	SSH_AVX2: TDES in ECB mode (d/e; KO1)	A 1677
	SSH_AVX: TDES in ECB mode (d/e; KO 1)	A 1678
	SSH_SSSE3: TDES in ECB mode (d/e; KO 1)	A 1679
	SSH_ASM: TDES in ECB mode (d/e; KO 1)	A 1680
Secure Hash Standard		
SHS	**SHA_AVX2** **SHA** (1, 224, 256, 384, 512)	A 1694
	SHA_AVX **SHA** (1, 224, 256, 384, 512)	A 1695
	SHA_SSSE3 **SHA** (1, 224, 256, 384, 512)	A 1696
	SHA_ASM **SHA** (1, 224, 256, 384, 512)	A 1697
	NEON: **SHA** (256)	A 2140
SHA-3	**SHA-3_AVX2** SHA3-224, SHA3-256, SHA3-384, SHA3-512, SHAKE-128, SHAKE-256	A 1681
	SHA-3_AVX512 SHA3-224, SHA3-256, SHA3-384, SHA3-512, SHAKE-128, SHAKE-256	A 1682
	SHA-3_ASM SHA3-224, SHA3-256, SHA3-384, SHA3-512, SHAKE-128, SHAKE-256	A 1683
Data Authentication Code		
HMAC	**SHA-3_AVX2** HMAC-SHA3-224, HMAC-SHA3-256, HMAC-SHA3-384, HMAC-SHA3-512	A 1681
	SHA-3_AVX512 HMAC-SHA3-224, HMAC-SHA3-256, HMAC-SHA3-384, HMAC-SHA3-512	A 1682
	SHA-3_ASM HMAC-SHA3-224, HMAC-SHA3-256, HMAC-SHA3-384, HMAC-SHA3-512	A 1683
	SHA_AVX2 **HMAC-SHA** (1, 224, 256, 384, 512)	A 1694
	SHA_AVX **HMAC-SHA** (1, 224, 256, 384, 512)	A 1695
	SHA_SSSE3 **HMAC-SHA** (1, 224, 256, 384, 512)	A 1696
	SHA_ASM **HMAC-SHA** (1, 224, 256, 384, 512)	A 1697

Approved or Allowed Security Functions		Cert #
	NEON: **HMAC-SHA** (256)	A 2140
Asymmetric Algorithms		
RSA	**SHA_AVX2:** **FIPS186-4:** PKCS 1.5 (Key Gen); Modulus Sizes 2048, 3072, 4096 PKCS 1.5 (Sig Gen) Modulus Sizes 2048, 3072, 4096 with hash sizes SHA-224, SHA-256, SHA-384, SHA-512 PKCS 1.5 (Sig Ver) Modulus Sizes 1024, 2048, 3072, 4096 with hash sizes SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 PSS (Sig Gen) Modulus Sizes 2048, 3072, 4096 with hash sizes SHA-224, SHA-256, SHA-384, SHA-512 PSS (Sig Ver) Modulus Sizes 1024, 2048, 3072, 4096 with hash sizes SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 X9.31 (Sig Gen) Modulus Sizes 2048, 3072, 4096 with hash sizes SHA-256, SHA-384, SHA-512 X9.31 (Sig Ver) Modulus Sizes 1024, 2048, 3072, 4096 with hash sizes SHA-1, SHA-256, SHA-384, SHA-512.	A 1694
	SHA_AVX: **FIPS186-4:** PKCS 1.5 (Key Gen); Modulus Sizes 2048, 3072, 4096 PKCS 1.5 (Sig Gen) Modulus Sizes 2048, 3072, 4096 with hash sizes SHA-224, SHA-256, SHA-384, SHA-512 PKCS 1.5 (Sig Ver) Modulus Sizes 1024, 2048, 3072, 4096 with hash sizes SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 PSS (Sig Gen) Modulus Sizes 2048, 3072, 4096 with hash sizes SHA-224, SHA-256, SHA-384, SHA-512 PSS (Sig Ver) Modulus Sizes 1024, 2048, 3072, 4096 with hash sizes SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 X9.31 (Sig Gen) Modulus Sizes 2048, 3072, 4096 with hash sizes SHA-256, SHA-384, SHA-512 X9.31 (Sig Ver) Modulus Sizes 1024, 2048, 3072, 4096 with hash sizes SHA-1, SHA-256, SHA-384, SHA-512.	A 1695
	SHA_SSSE3: **FIPS186-4:** PKCS 1.5 (Key Gen); Modulus Sizes 2048, 3072, 4096 PKCS 1.5 (Sig Gen) Modulus Sizes 2048, 3072, 4096 with hash sizes SHA-224, SHA-256, SHA-384, SHA-512 PKCS 1.5 (Sig Ver) Modulus Sizes 1024, 2048, 3072, 4096 with hash sizes SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 PSS (Sig Gen) Modulus Sizes 2048, 3072, 4096 with hash sizes SHA-224, SHA-256, SHA-384, SHA-512 PSS (Sig Ver) Modulus Sizes 1024, 2048, 3072, 4096 with hash sizes SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	A 1696

Approved or Allowed Security Functions		Cert #
	X9.31 (Sig Gen) Modulus Sizes 2048, 3072, 4096 with hash sizes SHA-256, SHA-384, SHA-512 X9.31 (Sig Ver) Modulus Sizes 1024, 2048, 3072, 4096 with hash sizes SHA-1, SHA-256, SHA-384, SHA-512.	
	SHA_ASM: **FIPS186-4:** PKCS 1.5 (Key Gen); Modulus Sizes 2048, 3072, 4096 PKCS 1.5 (Sig Gen) Modulus Sizes 2048, 3072, 4096 with hash sizes SHA-224, SHA-256, SHA-384, SHA-512 PKCS 1.5 (Sig Ver) Modulus Sizes 1024, 2048, 3072, 4096 with hash sizes SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 PSS (Sig Gen) Modulus Sizes 2048, 3072, 4096 with hash sizes SHA-224, SHA-256, SHA-384, SHA-512 PSS (Sig Ver) Modulus Sizes 1024, 2048, 3072, 4096 with hash sizes SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 X9.31 (Sig Gen) Modulus Sizes 2048, 3072, 4096 with hash sizes SHA-256, SHA-384, SHA-512 X9.31 (Sig Ver) Modulus Sizes 1024, 2048, 3072, 4096 with hash sizes SHA-1, SHA-256, SHA-384, SHA-512.	A 1697
DSA	**SHA_AVX2** **FIPS186-4:** Key Gen, PQG Gen, PQG Ver, Sig Gen, Sig Ver; Modulus Sizes 2048, 3072, with hash sizes SHA-224, SHA-256, SHA-384, SHA-512; (Modulus size 1024 and SHA-1 allowed for Sig Ver operations only)	A 1694
	SHA_AVX **FIPS186-4:** Key Gen, PQG Gen, PQG Ver, Sig Gen, Sig Ver; Modulus Sizes 2048, 3072, with hash sizes SHA-224, SHA-256, SHA-384, SHA-512; (Modulus size 1024 and SHA-1 allowed for Sig Ver operations only)	A 1695
	SHA_SSSE3: **FIPS186-4:** Key Gen, PQG Gen, PQG Ver, Sig Gen, Sig Ver; Modulus Sizes 2048, 3072, with hash sizes SHA-224, SHA-256, SHA-384, SHA-512; (Modulus size 1024 and SHA-1 allowed for Sig Ver operations only)	A 1696
	SHA_ASM **FIPS186-4:** Key Gen, PQG Gen, PQG Ver, Sig Gen, Sig Ver; Modulus Sizes 2048, 3072, with hash sizes SHA-224, SHA-256, SHA-384, SHA-512; (Modulus size 1024 and SHA-1 allowed for Sig Ver operations only)	A 1697
ECDSA[1]	**SHA_AVX2:** **FIPS186-4:**	A 1694

[1] Please note that ECDSA is also tested on the CAVP cert A1683, however it only corresponds to the prerequisite SHA3 assembly implementation and the ECDSA implementation itself is the same as what is tested with rest of the ACVP certificates listed in this entry.

Approved or Allowed Security Functions		Cert #
	Key Gen, Key Ver, Sig Gen, Sig Ver; Curves P-256, P-384, P-521 with hash sizes SHA-224, SHA-256, SHA-384, SHA-512; (SHA-1 allowed for Sig Ver operations only) (Curve P-224 is only used with Sig Gen and Sig Ver)	
	SHA_AVX: **FIPS186-4:** Key Gen, Key Ver, Sig Gen, Sig Ver; Curves P-256, P-384, P-521 with hash sizes SHA-224, SHA-256, SHA-384, SHA-512; (SHA-1 allowed for Sig Ver operations only) (Curve P-224 is only used with Sig Gen and Sig Ver)	A 1695
	SHA_SSSE3: **FIPS186-4:** Key Gen, Key Ver, Sig Gen, Sig Ver; Curves P-256, P-384, P-521 with hash sizes SHA-224, SHA-256, SHA-384, SHA-512; (SHA-1 allowed for Sig Ver operations only) (Curve P-224 is only used with Sig Gen and Sig Ver)	A 1696
	SHA_ASM **FIPS186-4:** Key Gen, Key Ver, Sig Gen, Sig Ver; Curves P-256, P-384, P-521 with hash sizes SHA-224, SHA-256, SHA-384, SHA-512; (SHA-1 allowed for Sig Ver operations only) (Curve P-224 is only used with Sig Gen and Sig Ver)	A 1697
Random Number Generation (NIST SP 800-90Ar1)		
DRBG	**AESNI:** AES ctr DRBG [Prediction Resistance Tested: Enabled and Not Enabled; Supports Reseed: (AES-128, AES-192, AES-256)]	A 1673
	AESASM: AES ctr DRBG [Prediction Resistance Tested: Enabled and Not Enabled; Supports Reseed: (AES-128, AES-192, AES-256)]	A 1674
	BAES_CTASM: AES ctr DRBG [Prediction Resistance Tested: Enabled and Not Enabled; Supports Reseed: (AES-128, AES-192, AES-256)]	A 1675
	AES_C: AES ctr DRBG [Prediction Resistance Tested: Enabled and Not Enabled; Supports Reseed: (AES-128, AES-192, AES-256)]	A 2139
	CE: AES ctr DRBG [Prediction Resistance Tested: Enabled and Not Enabled; Supports Reseed: (AES-128, AES-192, AES-256)]	A 2141
	VPAES: AES ctr DRBG [Prediction Resistance Tested: Enabled and Not Enabled; Supports Reseed: (AES-128, AES-192, AES-256)]	A 2142
Key Agreement Scheme Using HMAC Key Derivation Function (SP 800-56Cr1)		
KDA HKDF	**HKDF 56Cr1 to support the TLS 1.3 PRF:** Fixed Info Pattern: uPartyInfo||vPartyInfo Fixed Info Encoding: concatenation Derived Key Length: 2048 Shared Secret Length: 224-65336 Increment 8 HMAC Algorithm: SHA2-224, SHA2-256, SHA2-384, SHA2-512	A 1676
Key Agreement (NIST SP 800-56Ar3)		

Approved or Allowed Security Functions		Cert #
KAS-FFC-SSC SP 800-56Ar3	**FFC_DH:** Domain Parameter Generation and Mod P Methods (ffdhe2048, ffdhe3072, ffdhe4096, ffdhe6144, ffdhe8192, MODP-2048, MODP-3072, MODP-4096, MODP-6144, MODP-8192) Scheme: dhEphem KAS role: initiator, responder	A 1699
KAS-ECC-SSC SP 800-56Ar3	**SHA_AVX2:** Domain Parameter Generation Methods: (Curves P-256, P-384, P-521). Scheme: ephemeralUnified KAS role: initiator, responder	A 1694
	SHA_AVX: Domain Parameter Generation Methods: (Curves P-256, P-384, P-521). Scheme: ephemeralUnified KAS role: initiator, responder	A 1695
	SHA_SSSE3: Domain Parameter Generation Methods: (Curves P-256, P-384, P-521). Scheme: ephemeralUnified KAS role: initiator, responder	A 1696
	SHA_ASM: Domain Parameter Generation Methods: (Curves P-256, P-384, P-521). Scheme: ephemeralUnified KAS role: initiator, responder	A 1697
Safe Primes Key Generation and Verification	**FFC_DH:** Safe Prime Groups: ffdhe2048, ffdhe3072, ffdhe4096, ffdhe6144, ffdhe8192, MODP-2048, MODP-3072, MODP-4096, MODP-6144, MODP-8192	A 1699
KAS	KAS-ECC-SSC SP 800-56Ar3 with P-256, P-384, P-521; KAS-FFC-SSC SP 800-56Ar3 with ffdhe2048, ffdhe3072, ffdhe4096, ffdhe6144, ffdhe8192; KDF TLS v 1.0/1.1, v1.2	A 1694 (KAS-SSC) A 1695 (KAS-SSC) A 1696 (KAS-SSC) A 1697 (KAS-SSC) A 1699 (KAS-SSC) A 1694 (CVL) A 1695 (CVL) A 1696 (CVL) A 1697 (CVL)
RSA Key Transport Scheme (SP 800-56Br2)		
KTS-IFC	**SHA_AVX2:** Function: partialVal; Modulo: 2048, 3072, 4096, 6144, 8192; Key Generation Methods: rsakpg1-basic; Scheme: KTS-OAEP-basic: KAS Role: initiator, responder; Key Transport Method Hash Algorithms: SHA2-224, SHA2-256, SHA2-384, SHA2-512, SHA3-224, SHA3-256, SHA3-384, SHA3-512	A 1694
	SHA_AVX: Function: partialVal; Modulo: 2048, 3072, 4096, 6144, 8192; Key Generation Methods: rsakpg1-basic; Scheme: KTS-OAEP-basic:	A 1695

Approved or Allowed Security Functions		Cert #
	KAS Role: initiator, responder; Key Transport Method Hash Algorithms: SHA2-224, SHA2-256, SHA2-384, SHA2-512, SHA3-224, SHA3-256, SHA3-384, SHA3-512	
	SHA_SSSE3: Function: partialVal; Modulo: 2048, 3072, 4096, 6144, 8192; Key Generation Methods: rsakpg1-basic; Scheme: KTS-OAEP-basic: KAS Role: initiator, responder; Key Transport Method Hash Algorithms: SHA2-224, SHA2-256, SHA2-384, SHA2-512, SHA3-224, SHA3-256, SHA3-384, SHA3-512	A 1696
	SHA_ASM: Function: partialVal; Modulo: 2048, 3072, 4096, 6144, 8192; Key Generation Methods: rsakpg1-basic; Scheme: KTS-OAEP-basic: KAS Role: initiator, responder; Key Transport Method Hash Algorithms: SHA2-224, SHA2-256, SHA2-384, SHA2-512, SHA3-224, SHA3-256, SHA3-384, SHA3-512	A 1697
Key Derivation Using Pseudo Random Functions (SP 800-108)		
KBKDF	**KBKDF:** KDF Mode: Counter and Feedback MAC Modes: HMAC-SHA-1, HMAC-SHA2-224, HMAC-SHA2-256, HMAC-SHA2-384, HMAC-SHA2-512, CMAC-AES128, CMAC-AES192, CMAC-AES256, CMAC-TDES	A 1698
Transport Layer Security Key Derivation Function (SP 800-135)		
KDF-TLS (CVL)	**SHA_AVX2** (TLS 1.0, TLS 1.1, TLS 1.2, (SHA2 256, SHA2 384)	A 1694
	SHA_AVX (TLS 1.0, TLS 1.1, TLS 1.2, (SHA2 256, SHA2 384)	A 1695
	SHA_SSSE3: (TLS 1.0, TLS 1.1, TLS 1.2, (SHA2 256, SHA2 384)	A 1696
	SHA_ASM (TLS 1.0, TLS 1.1, TLS 1.2, (SHA2 256, SHA2 384)	A 1697
SSH Key Derivation Function (SP 800-135)		
KDF-SSH (CVL)	**SSH_ASM:** Cipher: Triple-DES Hash Algorithm: SHA-1, SHA-256, SHA-384, SHA-512 Cipher: AES-128, AES-192, AES-256 Hash algorithm: SHA-1, SHA-256, SHA-384, SHA-512	A 1680 A 2143[2]
	SSH_SSSE3: Cipher: Triple-DES Hash Algorithm: SHA-1, SHA-256, SHA-384, SHA-512 Cipher: AES-128, AES-192, AES-256 Hash algorithm: SHA-1, SHA-256, SHA-384, SHA-512	A 1679

[2] This certificate also lists testing of ECB mode for AES and Triple-DES algorithms which is the same C implementation tested under certificate #A1672 for Triple-DES and #A2139 for AES.

Approved or Allowed Security Functions		Cert #
	SSH_AVX2: Cipher: Triple-DES Hash Algorithm: SHA-1, SHA-256, SHA-384, SHA-512 Cipher: AES-128, AES-192, AES-256 Hash algorithm: SHA-1, SHA-256, SHA-384, SHA-512	A 1677
	SSH_AVX: Cipher: Triple-DES Hash Algorithm: SHA-1, SHA-256, SHA-384, SHA-512 Cipher: AES-128, AES-192, AES-256 Hash algorithm: SHA-1, SHA-256, SHA-384, SHA-512	A 1678
Password Based Key Derivation Function		
PBKDF2	**SHA_AVX2** Hash Algorithm: SHA1, SHA-224, SHA-256, SHA-384, SHA-512	A 1694
	SHA_AVX Hash Algorithm: SHA1, SHA-224, SHA-256, SHA-384, SHA-512	A 1695
	SHA_SSSE3 Hash Algorithm: SHA1, SHA-224, SHA-256, SHA-384, SHA-512	A 1696
	SHA_ASM Hash Algorithm: SHA1, SHA-224, SHA-256, SHA-384, SHA-512	A 1697
	SHA-3_AVX2 Hash Algorithm: SHA3-224, SHA3-256, SHA3-384, SHA3-512	A 1681
	SHA-3_AVX512 Hash Algorithm: SHA3-224, SHA3-256, SHA3-384, SHA3-512	A 1682
	SHA-3_ASM Hash Algorithm: SHA3-224, SHA3-256, SHA3-384, SHA3-512	A 1683
Key Transport Scheme (KTS)		
KTS	**AESNI:** AES-KW (D/E; Key Sizes 128 , 192, 256) AES-KWP (D/E; Key Sizes 128, 192, 256)	A 1673
	AESASM: AES-KW (D/E; Key Sizes 128 , 192, 256) AES-KWP (D/E; Key Sizes 128, 192, 256)	A 1674
	BAES_CTASM: AES-KW (D/E; Key Sizes 128 , 192, 256) AES-KWP (D/E; Key Sizes 128, 192, 256)	A 1675
	AES_C: AES-KW (D/E; Key Sizes 128 , 192, 256) AES-KWP (D/E; Key Sizes 128, 192, 256)	A 2139
	CE: AES-KW (D/E; Key Sizes 128 , 192, 256) AES-KWP (D/E; Key Sizes 128, 192, 256)	A 2141
	VPAES: AES-KW (D/E; Key Sizes 128 , 192, 256) AES-KWP (D/E; Key Sizes 128, 192, 256)	A 2142
	AES-GCM key wrapping with 128, 192, 256 bit keys	A 1685

Approved or Allowed Security Functions		Cert #
		A 1686 A 1687 A 1688 A 1689 A 1690 A 1691 A 1692 A 1693 A 2144 A 2145 A 2146
	AES-CCM key wrapping with 128, 192, 256 bit keys	A 1673 A 1674 A 1675 A 2139 A 2141 A 2142
	AES-CBC with 128 and 256 bit keys and HMAC-SHA-1, HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384, or HMAC-SHA-512 key wrapping.	A 1673 (AES) A 1674 (AES) A 1675 (AES) A 2139 (AES) A 2141 (AES) A 2142 (AES) A 1694 (HMAC) A 1695 (HMAC) A 1696 (HMAC) A 1697 (HMAC) A 2140 (HMAC-SHA2-256)
	Triple-DES CBC with 192 bit key and HMAC-SHA-1, HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384, or HMAC-SHA-512 key wrapping.	A 1672 (Triple-DES) A 1694 (HMAC) A 1695 (HMAC) A 1696 (HMAC) A 1697 (HMAC) A 2140 (HMAC-SHA2-256)
Entropy (NIST SP 800-90B)		
ENT (NP)	**NIST SP 800-90B**	N/A

Table 2: FIPS Approved or Allowed Security Functions

Note: No parts of the TLS protocol except the KDF with certs #A 1694 , #A 1695 , #A 1696 and #A 1697 have been reviewed or tested by CMVP or CAVP. Also, no parts of the SSH protocol except the KDF with certs #A 1677, #A 1678, #A 1679, #A 1680 and #A 2143 have been reviewed or tested by CMVP or CAVP.

3.4 Non-Approved But Allowed Security Functions

The following are considered non-Approved but allowed security functions provided by the Module:

Algorithm	Usage
RSA PKCS#1-v1.5 Key Wrapping with key sizes greater than 2048-bit up to 16384 bits	Key wrapping, key establishment methodology provides between 112 and 256 bits of encryption strength, non-compliant less than 112 bits.
MD5 (no security claimed per IG 1.23)	Message digest used in TLS only

Table 3: Non-Approved but Allowed Security Functions

3.5 Non-Approved Security Functions

Security functions listed in the table below, make use of non-approved cryptographic algorithms. Use of any of these algorithms and services in Table 9 will put the module in the non-Approved mode implicitly. The services associated with these algorithms are specified in section 7.2.

Algorithm	Usage
AES-OCB	Authenticated Encryption/Decryption
ANSI X9.31 RNG	Random Number Generation
ARIA	Encryption/Decryption
BLAKE2	Hash function
Blowfish	Encryption/Decryption
Camellia	Encryption/Decryption
CAST	Encryption/Decryption
CAST5	Encryption/Decryption
ChaCha20	Encryption/Decryption
ChaCha20 and Poly1305	Authenticated Encryption/Decryption
DES	Encryption/Decryption
Diffie-Hellman with non-compliant key size	Key agreement and shared secret computation using primes not listed in Table 2
Diffie-Hellman keys generated with domain parameters other than safe primes.	Key agreement, Shared Secret computation
DSA	DSA key pair generation and domain parameter generation with key size less than 2048 bits or greater than 3072 bits.
	DSA signature generation with key size less than 2048 bits or greater than 3072 bits. DSA signature generation with SHA-1.
	DSA domain parameter generation/verification less than 2048 bits or greater than 3072 bits. DSA domain parameter generation/verification with SHA-1
	DSA signature verification with key size less than 1024 bits or greater than 3072 bits.
EC Diffie-Hellman with P-192 curve, K curves, B curves and non-NIST curves.	Key agreement, Shared Secret computation
ECDSA with P-192 curve, K curves, B curves and non-NIST curves.	Key generation/Key verification/Signature generation/Signature Verification with NIST curve P-192, K curves, B curves and non-NIST curves
	Signature generation with SHA-1

Hash_DRBG	Random Number Generation
HMAC_DRBG	Random Number Generation
HMAC with non-compliant key size	HMAC with less than 112-bit keys
IDEA	Encryption/Decryption
J-PAKE	Password Authenticated Key Exchange
KRB5KDF	Key Derivation
MD2	Hash Function
MD4	Hash Function
MDC2	Hash Function
RC2	Encryption/Decryption
RC4	Encryption/Decryption
RC5	Encryption/Decryption
RIPEMD	Hash Function
RSA	RSA key pair generation with key size less than 2048 bits or greater than 4096 bits.
	RSA Signature generation with SHA-1
	RSA signature generation with key size less than 2048 bits or greater than 4096 bits.
	RSA signature verification with key size less than 1024 bits or greater than 4096 bits.
	RSA key wrapping using less than 2048 bit keys
SEED	Encryption/Decryption
Single Step KDF	Key Derivation
Whirlpool	Hash Function

Table 4: Non-Approved Disallowed Functions

4. Module Ports and Interfaces

The module interfaces can be categorized as follows:

- Data Input Interface
- Data Output Interface
- Control Input interface
- Status Output Interface

The module can be accessed by utilizing the API function it provides. Table 5 below, shows the mapping of interfaces as per FIPS 140-2 Standard.

FIPS 140 Interface	Module Interfaces
Data Input	API Input Parameters
Data Output	API Output Parameters
Control Input	API Function Calls
Status Output	API Return Values, error message

Table 5: Mapping of FIPS 140 Logical Interfaces

5. Physical Security

The Module is comprised of software only and thus does not claim any physical security.

6. Operational Environment

6.1 Tested Environments

The Modules were tested on the following environments with and without PAA i.e. AES-NI:

Module Version	Operating Environment	Processor	Hardware
R8-8.4.0	Oracle Linux 8.4 64 bit	Intel® Xeon® 8167M	Oracle Server X7-2C
R8-8.4.0	Oracle Linux 8.4 64 bit	AMD EPYC™ 7551	Oracle Server E1-2C
R8-8.4.0	Oracle Linux 8.4 64-bit	Ampere®Altra® Neoverse-N1	Oracle Server A1-2C

Table 6: Tested Operating Environment

6.2 Vendor Affirmed Environments

The following platforms have not been tested as part of the FIPS 140-2 level 1 certification however Oracle "vendor affirms" that these platforms are equivalent to the tested and validated platforms. Additionally, Oracle affirms that the module will function the same way and provide the same security services on any of the systems listed below.

Operating Environment	Hardware
Oracle Linux 8 64-bit	Oracle X Series Servers
Oracle Linux 8 64-bit	Oracle E Series Servers
Oracle Linux 8 64-bit	Oracle A Series Servers
Oracle Linux 8 64-bit	Marvell CN23XX OCTEON (MIPS) SmartNIC
Oracle Linux 8 64-bit	Marvell CN93XX LiquidIO III (ARM) SmartNIC
Oracle Linux 8 64-bit	Pensando DSC-200 (ARM) SmartNIC

Table 7: Vendor Affirmed Operating Environment

CMVP makes no statement as to the correct operation of the module or the security strengths of the generated keys when so ported if the specific operational environment is not listed on the validation certificate.

6.3 Operational Environment Policy

The operating system is restricted to a single operator mode of operation (i.e., concurrent operators are explicitly excluded).

The application that makes calls to the Modules is the single user of the Modules, even when the application is serving multiple clients.

During module operation, the ptrace(2) system call, the debugger (gdb(1)), and strace(1) shall not be used. In addition, other tracing mechanisms offered by the Linux environment, such as Dtrace, ftrace or systemtap, shall not be used.

7. Roles, Services and Authentication

7.1 Roles

The Oracle Linux OpenSSL Cryptographic Module supports 2 roles as required by FIPS PUB 140-2. These roles are a Crypto Officer Role and a User Role. Both roles are implicitly assumed by the entity accessing services implemented by the Module. The module does not support authentication mechanisms.

7.2 FIPS Approved Operator Services and Descriptions

The below table provides a full description of FIPS Approved services provided by the module and lists the roles allowed to invoke each service. In the table below, the "U" represents a User Role, and "CO" denotes a Crypto Officer role.

U	CO	Service Name	Service Description	Keys and CSP(s)	Access Type(s)
X		Symmetric Encryption/Decryption	Encrypts or decrypts a block of data using AES or 3-Key Triple-DES	AES or 3-Key Triple-DES Key	R, X
X		Asymmetric Encryption/Decryption	Encrypts or decrypts using Approved RSA key size	RSA key pair keys listed in Table 2	R, X
X		Certificate Management Handling	Management of key properties within certificates.	RSA, DSA, and ECDSA private keys listed in Table 2	R, X
X		Asymmetric Key Generation	Generate RSA, DSA and ECDSA asymmetric keys	RSA, DSA and ECDSA keys listed in Table 2	R, W, X
X		EC Key Verification	Verify ECDSA or ECDH keys	ECDSA and ECDH key listed in Table 2	R, X
X		Digital Signature Generation and Verification	Sign and verify operations	RSA, DSA, and ECDSA keys listed in Table 2	R, W, X
X		Asymmetric domain parameter Generation/Verification	Generate and verify DSA domain parameters	DSA public and private keys listed in Table 2	R, W, X
X		KAS-FFC-SSC	Shared secret computation for Diffie-Hellman	Diffie-Hellman public and private keys, Shared secret	R, X
X		Diffie-Hellman key generation and verification using safe primes	Safe Primes Key Generation and Verification	Diffie-Hellman public and private keys	R, W, X
X		KAS-ECC-SSC	Shared secret computation for EC Diffie-Hellman	EC Diffie-Hellman public and private keys, Shared secret	R, X

U	CO	Service Name	Service Description	Keys and CSP(s)	Access Type(s)
X		Key wrapping	AES-KW KTS, AES-KWP KTS, AES-GCM KTS, AES-CBC + HMAC KTS, Triple-DES CBC + HMAC KTS	AES key, Triple-DES key, HMAC key	R, X
X		TLS Network Protocol	Provide data encryption and authentication over TLS network protocol	AES, Triple-DES, and HMAC keys listed in Table 2.	R, X
X		TLS Key Agreement	Negotiate a TLS key agreement secure channel	AES or Triple-DES key, RSA, DSA or ECDSA private key, HMAC Key, Diffie-Hellman and EC Diffie-Hellman Private keys listed in Table 2, TLS pre-master secret and master secret	R, X
X		Key Derivation	TLS KDF	Shared secret, TLS derived key	R, W, X
			SSH KDF	Shared secret, SSH-KDF derived key	
			HKDF	Shared secret, HKDF derived key	
			PBKDF	PBKDF password and PBKDF derived key	
			KBKDF	Key derivation key, KBKDF derived key	
X		Keyed Hash (HMAC)	Sign and or authenticate data using HMAC-SHA	HMAC Keys listed in Table 2	R, X
X		Keyed Hash (CMAC)	Encrypt and sign data using CMAC.	AES or 3-Key Triple-DES Key	R, X
X		Hash (SHS)	Hash a block of data.	None	N/A
X		Random Number Generation	Generate random numbers based on the NIST SP 800-90A Standard	Entropy input string, seed and internal State (V and Key values)	R, W, X
X		Show Status	Show status of the module state	None	N/A
X		Self-Test	Initiate power-on self-tests	None	N/A
X		Zeroize	Zeroize all critical security parameters	All keys and CSP's	Z
	X	Module Installation	Installation of the module	None	N/A

R – Read, W – Write, X – Execute, Z - Zeroize

Table 8: FIPS Approved Services and Descriptions

7.3 Non-FIPS Approved Services and Descriptions

The following table lists the non-Approved services available in non-FIPS mode. Security services listed in the table below, make use of non-approved cryptographic algorithms. Use of any of these services in Table 9 will put the module in the non-Approved mode implicitly. The algorithms associated with these services are specified in section 3.5.

U	CO	Service Name	Service Description	Keys	Access Type(s)
X		Asymmetric Encryption/Decryption	Encrypts or decrypts using non-Approved RSA key size as listed in Table 4	RSA key pair	R, X
X		Symmetric Encryption/Decryption	Encrypts or decrypts using non-Approved algorithms	AES OCB, AES XTS, ARIA, Blowfish, Camellia, CAST, CAST5, ChaCha20, DES, IDEA, RC2, RC4, RC5, SEED keys	R, X
X		AEAD using Chacha20 and Poly1305	Authenticated encryption or decryption using ChaCha20 and Poly1305	ChaCha20 key	R, X
X		Digital Signature Generation	Sign operations with RSA, DSA and ECDSA restrictions listed in Table 4	RSA key < 2048 and RSA key > 4096 DSA key < 2048 and DSA key > 3072 ECDSA with NIST curve P-192, K curves, B curves and non-NIST curves Signature Generation with SHA-1	R, X
X		Digital Signature Verification	Verify operations with RSA, DSA and ECDSA restrictions listed in Table 4	RSA key < 1024 and RSA key > 4096 DSA key < 1024 and DSA key > 3072 ECDSA with NIST curve P-192, K curves, B curves and non-NIST curves	R, X
X		TLS Key Agreement	Negotiate a TLS key agreement secure channel with non-Approved key sizes	RSA/ Diffie-Hellman key < 2048 EC Diffie-Hellman with P-192 DSA/ECDSA key restrictions listed in Table 4	R, X
X		Key Derivation	Single Step KDF	SSKDF derived key	R, W, X
X		Asymmetric Key Generation	Generation of non-Approved RSA, DSA and ECDSA keys	RSA key < 2048 DSA key < 2048 ECDSA using NIST P-192 curve, K curves, B curves and non-NIST curves.	R, W, X
X		Random Number Generation	Generation of random numbers using the ANSI X9.31 PRNG	seed, seed key	R, W, X
			Generation of random numbers using the hash_drbg	Entropy input string, seed and internal State (V and C values)	
			Generation of random numbers using the HMAC_drbg	Entropy input string, seed and internal State (V and Key values)	
X		Keyed Hash (HMAC)	HMAC restriction listed in Table 4	HMAC with less than 112-bit keys	R, X
X		Hash	Hashing using non-Approved hash functions that include BLAKE2, MD2, MD4, MD5, MDC2, RIPEMD, Whirlpool	None	N/A

U	CO	Service Name	Service Description	Keys	Access Type(s)
X		J-PAKE Key Agreement	Password authenticated key agreement using J-PAKE	J-PAKE key pair	R, X
X		Diffie-Hellman non-compliant key size and shared secret computation	Diffie-Hellman restrictions listed in Table 4	Diffie-Hellman public and private keys	R, W, X
X		EC Diffie-Hellman shared secret computation	EC Diffie-Hellman restrictions listed in Table 4	EC Diffie-Hellman public and private keys	R, W, X

Table 9: Non-FIPS Approved Services and Descriptions

7.4 Operator Authentication

The module does not support operator authentication mechanisms.

8. Key and CSP Management

The following keys, cryptographic key components and other critical security parameters are contained in the module.

CSP Name	Generation/Input	Use	Zeroization
AES Key (128, 192, 256 bits)	The Key is passed into the module via API input parameter	Encrypt/Decrypt operations Used to generate and verify MAC's with AES as part of the CMAC algorithm.	EVP_CIPHER_CTX_free()
Triple-DES Keys (192 bits)		Used for Encrypt/Decrypt operations. Used for generating and verifying MAC's with Triple-DES as part of the CMAC algorithm.	
HMAC Key (≥ 112 bits)		HMAC keys used to generate and verify MAC's on data.	HMAC_CTX_cleanup()
RSA Key pair (2048, 3072, 4096 bits)	Keys are generated using FIPS 186-4 and the random value used in the key generation is generated using SP800- 90A DRBG	RSA public/private keys used to sign and verify data. RSA private key used for key decryption as part of key wrapping.	RSA_free()
DSA Key pair (2048, 3072, 4096 bits)		DSA public/private keys used to sign and verify data.	DSA_free()
ECDSA Key pair (P-224, P-256, P-384, P-521)		ECDSA public/private keys used to sign and verify data.	EC_KEY_free()
Diffie-Hellman Key pair	Public and private keys are generating using the SP 800-56Arev3 Safe Primes key generation method, random values are obtained from the SP800-90A DRBG.	Diffie-Hellman key pair used as part of the key agreement protocol.	DH_free()
EC Diffie-Hellman Key pair	Public and private keys are generated using the FIPS 186-4 key generation method, random values are obtained from the SP800 90A DRBG.	EC Diffie-Hellman key pair used as part of the key agreement protocol.	EC_KEY_free()
Shared secret	Generated during the Diffie-Hellman or EC Diffie-Hellman key agreement.	Used to derive the required keys by applying key derivation function.	DH_free(), EC_KEY_free()
TLS Pre-Master Secret	Established during the TLS handshake.	Entry: if received by module as TLS server, wrapped with server's public RSA key; otherwise no entry. Output: if generated by module as TLS client, wrapped with server's public RSA key; otherwise, no output.	SSL_free(), SSL_clear()
TLS Master Secret	Derived from TLS pre-master secret By using key derivation	Master secret is never output or entered	SSL_free(), SSL_clear()
TLS derived key	Generated during TLS KDF.	Derive the master secret from the pre-master secret, and to derive the session keys from the master secret	SSL_free(), SSL_clear()

CSP Name	Generation/Input	Use	Zeroization
HKDF derived key	Generated during HKDF.	Derived SP800-56Crev1 HKDF KDF mechanisms.	kdf_hkdf_free()
SSH-KDF derived key	Generated during SSH KDF.	Derived SP800-135 SSH KDF mechanisms.	kdf_sshkdf_free()
KBKDF derived key	Generated during KBKDF.	Derived SP800-108 KBKDF KDF mechanisms.	kbkdf_free()
PBKDF derived key	Generated during PBKDF.	Derived SP800-132 PBKDF mechanisms.	kdf_pbkdf2_free()
PBKDF Password	N/A	Input password used for PBKDF function.	kdf_pbkdf2_free()
Key derivation key for KBKDF	N/A	Input key used for KBKDF function.	kbkdf_free()
DRBG Entropy input string	Obtained from CPU jitter source.	Entropy input strings used as part of the DRBG process.	FIPS_drbg_free()
DRBG seed, Internal state (V, and Key value)	Derived from entropy input during DRBG initialization.	The seed is used as input into the DRBG mechanism. V and key are used as part of the CTR DRBG process.	FIPS_drbg_free()

Table 10: CSP Table

8.1 Random Number Generation

The module provides an SP 800-90A-compliant Deterministic Random Bit Generator (DRBG) for creation of key components of asymmetric keys, and random number generation.

The seeding (and automatic reseeding) of the DRBG is done with getrandom().

The module employs the Deterministic Random Bit Generator (DRBG) based on [SP 800-90A] for the random number generation. The DRBG supports the CTR_DRBG mechanisms. The module performs the DRBG health tests as defined in section 11.3 of [SP 800-90A]. The module uses CPU jitter as a noise source provided by the operational environment which is within the module's physical boundary but outside of the module's logical boundary. The source is compliant with [SP 800-90B] and marked as ENT on the certificate. The entropy provided from the entropy source provides 230 bits of entropy and the module requires up to 256 bits of entropy. The caveat, "The module generates cryptographic keys whose strengths are modified by available entropy" applies.

8.2 Key Generation

For generating RSA, DSA and ECDSA keys, the module implements asymmetric key generation services compliant with [FIPS 186-4] and using DRBG compliant with [SP 800-90A]. A seed (i.e. the random value) used in asymmetric key generation is obtained from [SP 800-90A] DRBG. In accordance with FIPS 140-2 IG D.12, the cryptographic module performs Cryptographic Key Generation (CKG) for asymmetric keys as per NIST SP 800-133. The resulting symmetric key or asymmetric seed is an unmodified output from a DRBG.

The public and private keys used in the EC Diffie-Hellman key agreement schemes are generated internally by the module using the ECDSA key generation method compliant with [FIPS186-4] and [SP 800-56Arev3]. The Diffie-

Hellman key agreement scheme is also compliant with [SP 800-56Arev3], and generates keys using safe primes defined in RFC7919 and RFC3526.

8.3 Key/CSP Storage

Public and private keys are provided to the Module by the calling process, and are destroyed when released by the appropriate API function calls. The Module does not perform persistent storage of keys.

8.4 Key/CSP Zeroization

The memory occupied by keys is allocated by regular memory allocation operating system calls. The application is responsible for calling the appropriate zeroization functions provided in the module's API and listed in Table 10. Calling the SSL_free() and SSL_clear() will zeroize the keys and CSPs stored in the TLS protocol internal state and also invoke the corresponding API functions listed in Table 10 to zeroize keys and CSPs. The zeroization functions overwrite the memory occupied by keys with "zeros" and deallocate the memory with the regular memory deallocation operating system call. In case of abnormal termination, or swap in/out of a physical memory page of a process, the keys in physical memory are overwritten by the Linux kernel before the physical memory is allocated to another process.

8.5 Key Establishment

The module provides KAS-FFC-SSC and KAS-ECC-SSC compliant with SP 800-56Arev3, in accordance with scenario X1 (1) of IG D.8.

For Diffie-Hellman, the module supports the use of safe primes from RFC7919 for domain parameters and key generation, which are used in the TLS key agreement implemented by the module.

- TLS (RFC7919)
  - ffdhe2048 (ID = 256)
  - ffdhe3072 (ID = 257)
  - ffdhe4096 (ID = 258)
  - ffdhe6144 (ID = 259)
  - ffdhe8192 (ID = 260)

The module also supports the use of safe primes from RFC3526, which are part of the Modular Exponential (MODP) Diffie-Hellman groups that can be used for Internet Key Exchange (IKE). Note that the module only implements key generation and verification, and shared secret computation using safe primes, but no part of the IKE protocol.

- IKEv2 (RFC3526)
  - MODP-2048 (ID=14)
  - MODP-3072 (ID=15)
  - MODP-4096 (ID=16)
  - MODP-6144 (ID=17)
  - MODP-8192 (ID=18)

Additionally, the module provides Diffie-Hellman and EC Diffie-Hellman key agreement schemes compliant with SP 800-56rev3 and used as part of the TLS protocol key exchange in accordance with scenario X1 (2) of IG D.8; that is, the shared secret computation (KAS-FFC-SSC and KAS-ECC-SSC) followed by the derivation of the keying material using SP 800-135 KDF.

According to Table 2: Comparable strengths in [SP 800-57], the key sizes of AES, Triple-DES, RSA, Diffie-Hellman and EC Diffie-Hellman provide the following security strength in FIPS mode of operation:

- KAS-FFC-SSC provides between 112 and 200 bits of encryption strength.
- KAS-ECC-SSC provides between 128 and 256 bits of encryption strength.
- Diffie-Hellman key agreement with TLS KDF provides between 112 and 200 bits of encryption strength.
- EC Diffie-Hellman key agreement with TLS KDF provides between 128 and 256 bits of encryption strength.
- RSA key wrapping with PKCS#1-v1.5 provides between 112 and 256 bits of encryption strength; Allowed per IG D.9
- AES key wrapping with KW and KWP key establishment methodology provides between 128 and 256 bits of encryption strength.
- AES key wrapping with CCM and GCM key establishment methodology provides between 128 and 256 bits of encryption strength.
- AES key wrapping with AES CBC and HMAC key establishment methodology provides 128 or 256 bits of encryption strength.
- Triple-DES key wrapping with Triple-DES CBC and HMAC key establishment methodology provides 112 bits of encryption strength.
- RSA key wrapping with OAEP key establishment methodology provides between 112 and 200 bits of encryption strength.

8.6 Key Derivation

The module supports the following key derivation method according to [SP 800-135]:

- KDF for the TLS protocol, used as pseudo-random functions (PRF) for TLSv1.0/1.1 and TLSv1.2
- KDF for the SSHv2 protocol

The module supports the following key derivation methods according to [SP 800-56C rev1]:

- KDF for the TLS protocol TLSv1.3.

The module supports the following key derivation methods according to [SP 800-108]:

- KBKDF for deriving a key from repeated application of a keyed MAC to an input secret (and other optional values).

The module also supports password-based key derivation (PBKDF). The implementation is compliant with option 1a of [SP 800-132]. Keys derived from passwords or passphrases using this method can only be used in storage applications.

8.7 Key/CSP Entry and Output

The module does not support manual key entry or intermediate key generation key output. The keys are provided to the module via API input parameters in plaintext form and output via API output parameters in plaintext form.

This is allowed by [FIPS 140-2_IG] IG 7.7, according to the “CM Software to/from App Software via GPC INT Path” entry on the Key Establishment Table.

9. Self-Tests

FIPS 140-2 requires that the Module performs self-tests to ensure the integrity of the Module, and the correctness of the cryptographic functionality at start up. In addition, conditional tests are required during operational stage of the module. All of these tests are listed and described in this section. See section 10.3 for descriptions of possible self-test errors and recovery procedure.

9.1 Power-Up Self-Tests

The Module performs power-up self-tests automatically during loading of the module by making use of default entry point (DEP) and no operator intervention is required. The module is not available for use until successful completion of power-up self-tests. Hence input, output, or any cryptographic functions cannot be performed while the Module is executing self-tests. The integrity of the module binary is verified using a HMAC-SHA-256. The HMAC value is computed at build time and stored in the hmac file. The value is recalculated at runtime and compared against the stored value. If the comparison succeeds, then the remaining power-up self-test (consisting of the algorithm-specific Known Answer Tests) are performed. On successful completion of the power-up tests, the module becomes operational and crypto services are available. If any of the tests fails module transitions to error state and subsequent calls to the Module will fail - thus no further cryptographic operations will be possible.

Algorithm	Test
AES	KAT AES ECB mode with 128-bit key, encryption and decryption are tested separately. KAT AES CCM mode with 192-bit key, encryption and decryption are tested separately. KAT AES GCM mode with 256-bit key, encryption and decryption are tested separately. KAT AES XTS mode with 128 and 256 bit keys, encryption and decryption are tested separately.
Triple-DES	KAT Triple-DES ECB mode, encryption and decryption are tested separately.
DSA	PCT, sign and verify with L=2048 and SHA-256.
RSA	KAT RSA with 2048-bit key, PKCS#1 v1.5 scheme with SHA-256 signature generation tested separately. KAT RSA with 2048-bit key, PKCS#1 v1.5 scheme with SHA-256, signature verification tested separately. KAT RSA with 2048-bit key, PSS scheme and SHA-256, signature generation tested separately. KAT RSA with 2048-bit key, PSS scheme and SHA-256, signature verification tested separately. KAT RSA with 2048-bit key, OAEP public key encryption and private key decryption tested separately.
ECDSA	PCT ECDSA with P-256 and SHA-256, sign and verify
KAS-FFC-SSC	Primitive "Z" Computation KAT with 2048-bit key
KAS-ECC-SSC	Primitive "Z" Computation KAT with P-256 curve
TLS KDF	KAT with SHA-256
SSH KDF	KAT with SHA-256
PBKDF2	KAT with SHA-256
HKDF KDF	KAT with SHA-256
KBKDF KDF	KAT with SHA-256
SP 800-90A CTR_DRBG	KAT CTR_DRBG with AES with 128, 192 and 256-bit keys
SP 800-90A DRBG	Health test per section 11.3 of SP 800-90A DRBG specification
HMAC	(SHA-1, SHA-224, SHA-256, SHA-384, SHA-512) KAT
SHA	(1, 256, 512) KAT
SHA-3	SHA3-256, SHA3-512 KAT
SHAKE	SHAKE-128, SHAKE-256 KAT
CMAC	KAT AES CMAC with 128, 192 and 256 bit keys, MAC generation KAT Triple-DES CMAC, MAC generation
Module Integrity	HMAC-SHA-256

Table 11: Power-On Self-Tests

9.2 Conditional Self-Tests

Conditional tests are performed during operational state of the module when the respective crypto functions are used. If any of the conditional tests fails, module transitions to error state.

Algorithm	Test
DSA Key generation	Pairwise Consistency Test: signature generation and verification
ECDSA Key generation	Pairwise Consistency Test: signature generation and verification
RSA Key generation	Pairwise Consistency Test: signature generation and verification, encryption and decryption

Table 12: Conditional Self-Tests

9.3 On-Demand self-tests

The module provides the Self-Test service to perform self-tests on demand. On demand self-tests can be invoked by powering-off and reloading the module. This service performs the same cryptographic algorithm tests executed during power-up. During the execution of the on-demand self-tests, crypto services are not available, and no data output or input is possible

10. Crypto-Officer and User Guidance

This section provides guidance for the Cryptographic Officer and the User to maintain proper use of the module per FIPS 140-2 requirements.

10.1 Crypto-Officer Guidance

The version of the RPM containing the validated module is stated in section 3.1 above. The RPM package of the Module can be installed by standard tools recommended for the installation of Oracle packages on an Oracle Linux system (for example, yum, RPM, and the RHN remote management tool). The integrity of the RPM is automatically verified during the installation of the Module and the Crypto Officer shall not install the RPM file if the Oracle Linux Yum Server indicates an integrity error. The RPM files listed in section 3 are signed by Oracle and during installation; Yum performs signature verification which ensures as secure delivery of the cryptographic module. If the RPM packages are downloaded manually, then the CO should run 'rpm –K <rpm-file-name>' command after importing the builder's GPG key to verify the package signature. In addition, the CO can also verify the hash of the RPM package to confirm a proper download.

Recommended method

The system-wide cryptographic policies package (crypto-policies) contains a tool that completes the installation of cryptographic modules and enables self-checks in accordance with the requirements of Federal Information Processing Standard (FIPS) Publication 140-2. We call this step "FIPS enablement". The tool named fips-mode-setup installs and enables or disables all the validated FIPS modules and it is the recommended method to install and configure an Oracle Linux 8 system.

1. Install RPM file openssl-1.1.1g-15.el8_3.x86_64.rpm or openssl-1.1.1g-15.el8_3.aarch64.rpm
   # yum install openssl
2. Ensure that the system is registered with the unbreakable Linux Network (ULN) and that the OL8_X86_64_latest or OL8_aarch64_latest channel is enabled
   # yum-config-manager --enable ol8_latest
3. Switch the system to FIPS enablement in Oracle Linux 8:
   # fips-mode-setup --enable
   Setting system policy to FIPS
   FIPS mode will be enabled.
   Reboot the system for the setting to take effect.
4. Restart your system:
   # reboot
5. After the restart, you can check the current state:
   # fips-mode-setup --check
   FIPS mode is enabled.

Note: As a side effect of the enablement procedure the fips-mode-enable tool also changes the system wide cryptographic policy level to a level named "FIPS", this level helps applications by changing configuration defaults to approved algorithms.

FIPS enablement via environment variable

1. Open:
   /etc/bashrc
2. Set:
   export OPENSSL_FORCE_FIPS_MODE=1

3. Save and close.
4. Run:

```
# source /etc/bashrc
```

10.1.1 AES Hardware Acceleration Support and Manual Method

According to the OpenSSL FIPS 140-2 Security Policy, the OpenSSL module supports the AES-NI Intel processor instruction and ARM AES optimizations set as an approved cipher. Both architecture optimizations are used by the Module.

In case you configured a full disk encryption using AES, you may use the aforementioned optimizations for a higher performance compared to the software-only implementation.

Verify that your processor offers AES hardware acceleration by calling the following command:

```
cat /proc/cpuinfo | grep aes
```

If the command returns a list of properties, including the “aes” string, your CPU provides the AES hardware acceleration. If the command returns nothing, AES hardware acceleration is not supported.

The recommended method automatically performs all the necessary steps. The following steps can be done manually but are not recommended and are not required if the systems has been installed with the fips-mode-setup tool:

- Create a file named /etc/system-fips, the contents of this file are never checked
- Ensure to invoke the command ‘fips-finish-install --complete’ on the installed system
- Ensure that the kernel boot line is configured with the fips=1 parameter set
- Reboot the system

NOTE: If /boot or /boot/efi resides on a separate partition, the kernel parameter boot=<boot partition> must be supplied. The partition can be identified with the command "df | grep boot". For example:

```
$ df  | grep boot
```

Filesystem	1K-blocks	Used	Available	Use%	Mounted on
/dev/sda1	233191	30454	190296	14%	/boot

The partition of the /boot file system is located on /dev/sda1 in this example.

Therefore the parameter boot=/dev/sda1 needs to be appended to the kernel command line in addition to the parameter fips=1.

10.2 User Guidance

In order to run the module in FIPS mode, only the FIPS approved or allowed services listed in table 8 or the validated or allowed cryptographic algorithms/security functions listed in Table 2 and Table 3 should be used.

ENGINE_register_*, ENGINE_set_default_* and FIPS_mode_set(0) function calls are prohibited.

10.2.1 TLS and Diffie-Hellman

The TLS protocol implementation provides both server and client sides. In order to operate in FIPS mode, digital certificates used for server and client authentication shall comply with the restrictions of key size and message digest algorithms imposed by [SP 800-131A]. In addition, for Diffie-Hellman only the safe prime groups listed in RFC7919 are approved to be used in FIPS mode.

10.2.2 Random Number Generator

The OpenSSL API call of RAND_cleanup must not be used. This call will clean up the internal DRBG state. This call also replaces the DRBG instance with the non-FIPS Approved SSLeay Deterministic Random Number Generator when using the RAND_* API calls.

10.2.3 AES GCM IV

The IV generation method is user selectable and may be computed in the following way:

Conforming to IG A.5, scenario #1 (for TLS 1.2): Comply with the provision of a peer-to-peer industry standard. Specifically, following RFC 5288 for TLS. The counter portion of the IV is set by the module within its cryptographic boundary. When the IV exhausts the maximum number of possible values for a given session key, the first party, client, or server to encounter this condition will trigger a handshake to establish a new encryption key in accordance with RFC 5246.

It is the responsibility of the user to determine which method to use.

In case the Modules' power is lost and then restored, the key used for the AES GCM encryption/decryption shall be re-distributed.

10.2.4 AES-XTS Guidance

The length of a single data unit encrypted or decrypted with the AES-XTS shall not exceed 2^{20} AES blocks that is 16MB of data per AES-XTS instance. An XTS instance is defined in section 4 of SP 800-38E.

The AES-XTS mode shall only be used for the cryptographic protection of data on storage devices. The AES-XTS shall not be used for other purposes, such as the encryption of data in transit. The module implements the check to ensure that the two AES keys used in XTS-AES algorithm are not identical.

10.2.5 Triple-DES Keys

Data encryption using the same three-key Triple-DES key shall not exceed 2^{16} Triple-DES blocks (2GB of data), in accordance to SP 800-67 and IG A.13.

[SP 800-67] imposes a restriction on the number of 64-bit block encryptions performed under the same three-key Triple-DES key.

When the three-key Triple-DES is generated as part of a recognized IETF protocol, the module is limited to 2^{20} 64-bit data block encryptions. This scenario occurs in the following protocol:

- Transport Layer Security (TLS) versions 1.1, 1.2, 1.3, conformant with [RFC 5246]

In any other scenario, the module cannot perform more than 2^{16} 64-bit data block encryptions. The user is responsible for ensuring the module's compliance with this requirement.

10.2.6 RSA and DSA Keys

An application can enforce the key generation bit length restriction for RSA and DSA keys by setting the environment variable OPENSSL_ENFORCE_MODULUS_BITS. This environment variable ensures that 1024 bit keys cannot be generated.

10.3 Handling Self-Test Errors

The Module transitions to error state when any of self-tests or conditional tests fails. The application must be restarted to recover from these errors. Following are the error messages specific to self-test failure:

FIPS_R_FINGERPRINT_DOES_NOT_MATCH – The integrity verification check failed
FIPS_R_SELFTEST_FAILED – a known answer test failed
FIPS_R_TEST_FAILURE – a known answer test failed (RSA); pairwise consistency test failed (DSA)
FIPS_R_PAIRWISE_TEST_FAILED – a pairwise consistency test failed during EC/DSA or RSA key generation
FIPS_R_SELFTEST_FAILURE – the DRBG Health Test failed

These errors are reported through the regular ERR interface of the Module and can be queried by functions such as ERR_get_error(). See the OpenSSL manual page for the function description.

When the Module is in error state, output is inhibited and no crypto operations are available. Any calls to the crypto functions in error state will return error message: 'FATAL FIPS SELFTEST FAILURE' on stderr and the application is terminated with the abort() call.

The only way to recover from the error state is to reload the module and restart the application. If failures persist, the Module must be reinstalled. If downloading the software, make sure to verify the package hash to confirm a proper download.

10.4 Key Derivation Using SP 800-132 PBKDF

The module provides password-based key derivation (PBKDF), compliant with SP 800-132. The module supports option 1a from section 5.4 of [SP 800-132], in which the Master Key (MK) or a segment of it is used directly as the Data Protection Key (DPK).

In accordance to [SP 800-132], the following requirements shall be met.

- Derived keys shall only be used in storage applications. The Master Key (MK) shall not be used for other purposes. The length of the MK or DPK shall be of 112 bits or more.
- A portion of the salt, with a length of at least 128 bits, shall be generated randomly using the SP 800-90A DRBG
- The iteration count shall be selected as large as possible, as long as the time required to generate the key using the entered password is acceptable for the users. The minimum value shall be 1000.
- Passwords or passphrases, used as an input for the PBKDF, shall not be used as cryptographic keys.
- The length of the password or passphrase shall be of at least 20 characters, and shall consist of lower-case, upper-case and numeric characters. The probability of guessing the value is estimated to be $1/62^{20} = 10^{-36}$, which is less than 2^{-112}.

The calling application shall also observe the rest of the requirements and recommendations specified in [SP 800-132].

11. Mitigation of Other Attacks

RSA is vulnerable to timing attacks. In a setup where attackers can measure the time of RSA decryption or signature operations, blinding must be used to protect the RSA operation from that attack.

The API function of RSA_blinding_on turns blinding on for key rsa and generates a random blinding factor. The random number generator must be seeded prior to calling RSA_blinding_on.

Weak Triple-DES keys are detected as follows:

```
/* Weak and semi week keys as taken from
* %A D.W. Davies
* %A W.L. Price
* %T Security for Computer Networks
* %I John Wiley & Sons
* %D 1984
* Many thanks to smb@ulysses.att.com (Steven Bellovin) for the reference
* (and actual cblock values).
*/
#define NUM_WEAK_KEY 16
static const DES_cblock weak_keys[NUM_WEAK_KEY]={
/* weak keys */
        {0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01},
        {0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE},
        {0x1F,0x1F,0x1F,0x1F,0x0E,0x0E,0x0E,0x0E},
        {0xE0,0xE0,0xE0,0xE0,0xF1,0xF1,0xF1,0xF1},
        /* semi-weak keys */
        {0x01,0xFE,0x01,0xFE,0x01,0xFE,0x01,0xFE},
        {0xFE,0x01,0xFE,0x01,0xFE,0x01,0xFE,0x01},
        {0x1F,0xE0,0x1F,0xE0,0x0E,0xF1,0x0E,0xF1},
        {0xE0,0x1F,0xE0,0x1F,0xF1,0x0E,0xF1,0x0E},
        {0x01,0xE0,0x01,0xE0,0x01,0xF1,0x01,0xF1},
        {0xE0,0x01,0xE0,0x01,0xF1,0x01,0xF1,0x01},
        {0x1F,0xFE,0x1F,0xFE,0x0E,0xFE,0x0E,0xFE},
        {0xFE,0x1F,0xFE,0x1F,0xFE,0x0E,0xFE,0x0E},
        {0x01,0x1F,0x01,0x1F,0x01,0x0E,0x01,0x0E},
        {0x1F,0x01,0x1F,0x01,0x0E,0x01,0x0E,0x01},
        {0xE0,0xFE,0xE0,0xFE,0xF1,0xFE,0xF1,0xFE},
        {0xFE,0xE0,0xFE,0xE0,0xFE,0xF1,0xFE,0xF1}};
```

Please note that there is no weak key detection by default. The caller can explicitly set the DES_check_key to 1 or call DES_check_key_parity() and/or DES_is_weak_key() functions on its own.

Acronyms, Terms and Abbreviations

Term	Definition
AES	Advanced Encryption Standard
AVX	Advanced Vector Extensions
CAVP	Cryptographic Algorithm Validation Program
CMVP	Cryptographic Module Validation Program
CSE	Communications Security Establishment
CSP	Critical Security Parameter
DH	Diffie-Hellman
DHE	Diffie-Hellman Ephemeral
DRBG	Deterministic Random Bit Generator
ECDH	Elliptic Curve Diffie-Hellman
ECDSA	Elliptic Curve Digital Signature Algorithm
GPG	Gnu Privacy Guard
HKDF	HMAC-based Extract-and-Expand Key Derivation Function
HMAC	(Keyed) Hash Message Authentication Code
IKE	Internet Key Exchange
KAT	Known Answer Test
KBKDF	Key-Based Key Derivation Function
KDF	Key Derivation Function
NIST	National Institute of Standards and Technology
PAA	Processor Algorithm Acceleration
PBKDF	Password-Based Key Derivation Function
PCT	Pairwise Consistency Test
POST	Power On Self Test
PR	Prediction Resistance
PSS	Probabilistic Signature Scheme
PUB	Publication
SHA	Secure Hash Algorithm
SSC	Shared Secret Computation
SSH	Secure Shell
SSKDF	Single Step Key Derivation Function
SSSE3	Supplemental Streaming SIMD Extensions 3
UEK	Oracle Linux Unbreakable Enterprise Kernel
TLS	Transport Layer Security

Table 13: Acronyms

References

The FIPS 140-2 standard, and information on the CMVP, can be found at http://csrc.nist.gov/groups/STM/cmvp/index.html. More information describing the module can be found on the Oracle web site at https://www.oracle.com/linux/ .

This Security Policy contains non-proprietary information. All other documentation submitted for FIPS 140-2 conformance testing and validation is "Oracle - Proprietary" and is releasable only under appropriate non-disclosure agreements.

Document	Author	Title
FIPS PUB 140-2	NIST	FIPS PUB 140-2: Security Requirements for Cryptographic Modules
FIPS IG	NIST	Implementation Guidance for FIPS PUB 140-2 and the Cryptographic Module Validation Program
FIPS PUB 140-2 Annex A	NIST	FIPS 140-2 Annex A: Approved Security Functions
FIPS PUB 140-2 Annex B	NIST	FIPS 140-2 Annex B: Approved Protection Profiles
FIPS PUB 140-2 Annex C	NIST	FIPS 140-2 Annex C: Approved Random Number Generators
FIPS PUB 140-2 Annex D	NIST	FIPS 140-2 Annex D: Approved Key Establishment Techniques
DTR for FIPS PUB 140-2	NIST	Derived Test Requirements (DTR) for FIPS PUB 140-2, Security Requirements for Cryptographic Modules
NIST SP 800-67	NIST	Recommendation for the Triple Data Encryption Algorithm TDEA Block Cipher
FIPS PUB 197	NIST	Advanced Encryption Standard
FIPS PUB 198-1	NIST	The Keyed Hash Message Authentication Code (HMAC)
FIPS PUB 186-4	NIST	Digital Signature Standard (DSS)
FIPS PUB 180-4	NIST	Secure Hash Standard (SHS)
NIST SP 800-131Ar1	NIST	Recommendation for the Transitioning of Cryptographic Algorithms and Key Sizes
PKCS#1	RSA Laboratories	PKCS#1 v2.1: RSA Cryptographic Standard
RFC 5288	https://tools.ietf.org/html/rfc5288	AES Galois Counter Mode (GCM) Cipher Suites for TLS

Table 14: References